Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 11 of 11 returned.

☐ 1. Document ID: US 6658515 B1

L14: Entry 1 of 11 File: USPT Dec 2, 2003

DOCUMENT-IDENTIFIER: US 6658515 B1
TITLE: Background execution of universal serial bus transactions

Brief Summary Text (10):
To facilitate communication between the computer system and 127 peripheral devices over a common serial line, the USB specification defines transactions between a host in data communication with a plurality of devices over interconnects. The USB interconnect defines the manner in which the USB devices are connected to and communicate with the USB host controller. There is generally only one host on any USB system. A USB interface to the host computer system is referred to as the host controller. The host controller may be implemented in a combination of hardware, firmware, or software. USB devices are defined as (1) hubs, which provide additional attachment points to the USB, or (2) functions, which provide capabilities to the system; e.g., an ISDN connection, a digital joystick, or speakers. Hubs indicate the attachment or removal of a USB device in its per port status bit. The host determines if a newly attached USB device is a hub or a function and assigns a unique USB address to the USB device. All USB devices are accessed by a unique USB address. Each device additionally supports one or more endpoints with which the host may communicate.

Detailed Description Text (2):
Referring to FIG. 3, PC system 100 includes a microprocessor ("CPU") 105, for example, an Intel.RTM. Pentium.RTM. class microprocessor, having a processor 110 for handling integer operations and a coprocessor 115 for handling floating point operations. CPU 105 is coupled to cache 129 and memory controller 130 via CPU bus 191. System controller I/O trap 192 couples CPU bus 191 to local bus 120 and is generally characterized as part of a system controller such as Pico Power Vesuvious or an Intel.TM. Mobile Triton chip set. System controller I/O trap 192 can be programmed in a well-known manner to intercept a particular target address or address range.

Detailed Description Text (4):
A graphics controller 135 is coupled to local bus 120 and to a panel display screen 140. Graphics controller 135 is also coupled to a video memory 145 that stores information to be displayed on panel display 140. Panel display 140 is typically an active matrix or passive matrix liquid crystal display ("LCD") although other display technologies may be used as well. Graphics controller 135 can also be coupled to an optional external display or standalone monitor display 156 as shown in FIG. 3. One graphics controller that can be employed as graphics controller 135 is the Western Digital WD90C24A graphics controller.

☐ 2. Document ID: US 6629188 B1

L14: Entry 2 of 11 File: USPT Sep 30, 2003

DOCUMENT-IDENTIFIER: US 6629188 B1
TITLE: Circuit and method for prefetching data for a texture cache

Abstract Text (1):
A cache memory apparatus for graphics and other systems. The cache memory apparatus includes a cache memory having a first number of cache lines, each cache line addressable by a cache line address; a first plurality of storage elements coupled to a first address bus; and a second plurality of storage elements coupled to the first plurality of storage elements. The first plurality of storage elements holds a second number of cache line addresses, and the second plurality of storage elements holds a third number of cache line addresses.

Brief Summary Text (2):
The present invention relates in general to graphics systems, and in particular to methods and apparatus for prefetching cache lines in a graphics system.

Brief Summary Text (3):
The sophistication of the market for computer and video graphics and games has exploded over the last few years. The time when simple games such as "Pong" was a marketable product is far in the past. Today's garners and computer users expect realistic three dimensional (3-D) images, whether the images are of a football game, race track, or new home's interior. Accordingly, this appetite has focused designers' efforts to improving the quality of the images produced by graphics systems in computers and video games.

Brief Summary Text (5):
A monitor's maximum resolution is set by the number of pixels on its screen. In color monitors, each pixel is made up of a red, green and blue "dot" in close proximity to one another. By varying the intensity of the "dots", the color and brightness of the pixel can be changed. The more pixels on a screen, the more realistic an image will appear. For example, if a typical tire on a race car is represented on the screen by one pixel, that pixel will be black. A single black spec on a screen would not make for a very impressive tire. But if the tire is represented by many pixels, then details such as shape, hub caps, lug nuts can be seen, and the image is more convincing. To add more realism, a texture, for example tire tread, can be added. Where the rubber meets the road, an asphalt texture may be used.

Brief Summary Text (6):
These textures are stored in memory, and are retrieved as required by the graphics system. They may be two dimensional or three dimensional. Two dimensional textures are two dimensional images, and the dimensional coordinates are typically labeled either s and t, or u and v. In systems using a conventional bilinear filter, four pieces of texture information, referred to as texels, are used to determine the texel value, which is the texture information for one pixel. 16 bits is a common size for each texel. Alternately, texels may be 4, 8, 32, or any other integral number of bits in size. Three dimensional textures are sets of two dimensional textures, and the coordinates are usually labeled s, t, and r. Trilinear filtering is common in systems supporting three dimensional textures, and uses 8 texels to determine the texture information for one pixel.

h e b b g e e e f e fe ef b e

Brief Summary Text (8):
This texel information is stored in memory for fast access by the graphics controller. Preferably it would all be stored in memory on the same chip as the other graphics system elements, using fast circuitry, such as static random access memory (SRAM). But SRAMs are large, and have high operating currents, so the die area and power costs are prohibitive.

Brief Summary Text (9):
A conventional solution to the problem of making a fast but cost effective memory is to use an architecture type known as a memory hierarchy. The concept behind memory hierarchy is to use a smaller amount of SRAM, preferably on-chip, and have a larger memory off-chip using less expensive circuitry, such as dynamic random access memory (DRAM). This way, some data needed quickly by the graphics controller is readily available in the on-chip fast SRAM, while the bulk of the data waits in the DRAM. If the controller needs data that is not available in the SRAM, it can pull the data from the DRAM and overwrite existing data in the SRAM. In this system, the SRAM is known as the cache, and the DRAM is the main memory. Memory hierarchy systems using cache may be used for storing texels in graphics systems.

Brief Summary Text (17):
An inherent drawback to this memory hierarchy scheme becomes apparent when it is contemplated for use in a graphics system as described above. In the CPU requests data from the cache, and a cache miss occurs, the cache requests and receives data from the main memory for presentation to the CPU. Unfortunately, the main memory is much slower than the cache memory and the CPU, thus every cache miss leaves the CPU idle for many CPU clock cycles. This is referred to as cache latency.

Brief Summary Text (18):
But in graphics systems, such as those consistent with embodiments of the present invention, texels are required at the tremendous speeds calculated above. The CPU cannot wait for the cache to retrieve data. This would result in "jumpy" or jittery graphic images being displayed. Rather, another solution which eliminates this cache miss latency must be found.

Drawing Description Text (4):
FIG. 2 is a block diagram illustrating a graphics system including a texture cache subsystem;

Detailed Description Text (2):
The Graphics Subsystem

Detailed Description Text (3):
FIG. 2 illustrates one embodiment of a graphics subsystem consistent with the present invention. The graphics subsystem includes a graphics pipeline 230, a display 240, a central processing unit (CPU) 200, a main memory 210, a memory controller 250, and a texture cache subsystem 220 connected together by various buses as shown. The main memory 210 has a number of storage elements, each holding a two dimensional texture image. Graphics pipeline 230 uses the textures in memory 210 to display surface detail such as texture and shading on objects in the image on display 240. Specifically, graphics pipeline 230 receives graphics primitives and other data from the CPU via memory controller 250. The data used by the graphics pipeline 230 includes the triangle vertices, each vertex being specified by x, y, and z coordinates, R, G, B color data, and s and t (or s, t, and r) texture coordinates. During this data processing, graphics pipeline 230 provides texel addresses to, and receives texels from, the texture cache subsystem 220. The graphics pipeline performs various functions including geometry processing, fragment generation, hidden surface removal, and frame buffer display.

Detailed Description Text (5):

As discussed above, inefficiencies arise in this architecture because the texels required by the CPU 200 are not always available in the texture cache subsystem 220, and must be fetched from the main memory 210. This is a comparatively slow process, taking many clock cycles to complete. Since a steady steam of texel quads is required by the graphics pipeline 230, it is necessary that fetching from main memory 210 happens in such a way that the image rendering on display 240 is not stalled. This wait time caused by a cache miss is referred to as cache miss latency.

Detailed Description Text (8):
The memory controller 350 sends requests for data to the main memory 360, as instructed by the texture cache manager 310. The memory controller 350 provides required texels which are not in cache to the texture cache controller 330 for placement in the texture cache 370. The texture cache controller 330 receives addresses from the FIFO 320, data from the memory controller 350, and provides data, specifically texels, for the texture filter 340. The texture cache controller 330 retrieves data from texture cache 370 specified by addresses from FIFO 320, and stores data received from the memory controller 350 in the texture cache 370. The location where this data is stored is discussed below. The texture cache controller 330 supplies the texels from the texture cache 370 to the texture filter 340.

Detailed Description Text (14):
FIG. 4 is a block diagram representing the circuitry used by one embodiment of the present invention. The circuit uses extra cache lines and one level of indirection to at least mitigate the overlapping index problem. Texture cache controller 330 details are shown, namely line read table 450, line fetch table 470, and line fetch address 480. Also included are an issued request counter 410, request manager 440, as well as the texture cache manager 310, memory controller 350, main memory 360, FIFO 320, texture cache 370, and texture filter 340.

Detailed Description Text (27):
The request manager 440 ensures that a cache line update will not be performed by the memory controller 350 unless there is an available cache line in texture cache 370. If there is a cache miss, the texture cache manager 310 sends the address on line 490 to the request manager 440. The request manager checks the count provided on line 463. If the count is less than its maximum value, the request manager increments the issued request counter 410 by sending a signal on line 467, and passes the address to the memory controller, which then updates the cache line identified by the line fetch address 480. But if the count on bus 463 is at its maximum, the request manager does not send the address to the memory controller 350, rather it sends a stall signal on line 436 to the texture cache manager 310. The texture cache manager 310 then stalls the upstream processing. The downstream activity continues, and the FIFO 320 outputs addresses until there is a read table update, which decrements the issued request counter 410. This causes the count falls below its maximum, and the address is passed to the memory controller 310 by the request manager 440. The request manager 440 withdraws the instruction to stall, and the upstream processing begins.

Detailed Description Text (32):
Pre-fetch manager 1380 is similar to the request manager 440 above. Specifically, the pre-fetch manager 1380 periodically checks the count signal on bus 1308 from the issued request counter 410. For example, the pre-fetch manager 1380 may check every system clock cycle. Again, the count signal indicates the number of issued requests, as counted by the issued request counter 410. If the number of issued requests is less than the number of entries in the line fetch table, the pre-fetch manager 1380 polls the pending request FIFO 1310 for addresses by checking the status of the addresses available signal on line 1358. If there is an address in the pending request FIFO 1310, the pre-fetch manager 1380 passes the address on the output bus 1359 of the pending request FIFO 1310 to the memory controller 350 on bus 1363, and increments the issued request counter 410 by sending an increment

signal on line 1305. If the pending request FIFO 1310 is empty, the pre-fetch manager 1380 has no address to pass, and the memory controller does not perform a cache line update. But if the number of issued requests is equal to the number of entries in the line fetch table, the pre-fetch manager 1380 will not pass an address to the memory controller 350, even if there are addresses in the pending request FIFO 1310.

Detailed Description Text (37):
The fetched line counter is included to stall the downstream processing if required cache line updating has not occurred. The fetched line counter 460 is incremented each time the memory controller updates a cache line. When an updated cache line address moves from the line fetch table to the line read table, the fetched line counter is decremented. The line fetch counter 460 therefore indicates the number of entries in the line fetch table with active validity bits. When an address having an active fetch request is output from the FIFO 320, the texture cache controller checks to see if the fetched line count is equal or larger than the number of read table entries which the fetch request indicates must be updated. If it is, the read table is updated, the line fetch table is updated, and processing continues. But if the fetched line count is less than the number of read table entries that the fetch request indicates must be updated, downstream processing is stalled, more cache lines are updated until processing may resume.

Detailed Description Text (55):
If the fetch flag on line 1185 is active, the fetch flag indicates a cache miss has occurred, and the memory controller is instructed to retrieve from the main memory the data block having the same tag as the address on bus 300. Some number of cycles later, the data is present on bus 1190 and provided to the write handler 1150. The read/write synchronizer 1120 reads a data address entry from the second look-up table 1130, and provides the address to the write handler 1150. The write handler, having data on bus 1190 and an address on bus 1192, instructs the texture cache to write the data on bus 1190 in the cache line identified by the address on bus 1192.

Detailed Description Text (62):
FIG. 12 is a more complete block diagram of a texture cache controller and related circuitry as used by a system consistent with one embodiment of the present invention. Included are a memory controller 350, cache write handler 1150, read/write synchronizer 1120, line fetch table 470, cache manager 310, cache read handler 1140, texture cache 370, shown as four individual cache banks 0-3, texel sorter 1210, line read table 450, address unpacker 1220, and texture filter 340. Using four individual cache banks allows each bank to supply one texel on each clock cycle.

3. Document ID: US 6587898 B1

L14: Entry 3 of 11 File: USPT Jul 1, 2003

DOCUMENT-IDENTIFIER: US 6587898 B1
TITLE: Universal serial bus PC synchronization algorithm for peripheral devices

Brief Summary Text (10):
To facilitate communication between the computer system and 127 peripheral devices over a common serial line, the USB specification defines transactions between a

h e b b g e e e f e fe ef b e

host in data communication with a plurality of devices over interconnects. The USB interconnect defines the manner in which the USB devices are connected to and communicate with the USB host controller. There is generally only one host on any USB system. A USB interface to the host computer system is referred to as the host controller. The host controller may be implemented in a combination of hardware, firmware, or software. USB devices are defined as (1) hubs, which provide additional attachment points to the USB, or (2) functions, which provide capabilities to the system; e.g., an ISDN connection, a digital joystick, or speakers. Hubs indicate the attachment or removal of a USB device in its per port status bit. The host determines if a newly attached USB device is a hub or a function and assigns a unique USB address to the USB device. All USB devices are accessed by a unique USB address. Each device additionally supports one or more endpoints with which the host may communicate.

Detailed Description Text (2):
Referring to FIG. 3 PC system 100 includes a microprocessor ("CPU") 105, for example, an Intel.RTM. Pentium.RTM. class microprocessor, having a processor 110 for handling integer operations and a coprocessor 115 for handling floating point operations. CPU 105 is coupled to cache 129 and memory controller 130 via CPU bus 191. System controller I/O trap 192 couples CPU bus 191 to local bus 120 and is generally characterized as part of a system controller such as a Pico Power Vesuvious or an Intel.RTM. Mobile Triton chip set. System controller I/O trap 192 can be programmed in a well-known manner to intercept a particular target address or address range.

Detailed Description Text (4):
A graphics controller 135 is coupled to local bus 120 and to a panel display screen 140. Graphics controller 135 is also coupled to a video memory 145 that stores information to be displayed on panel display 140. Panel display 140 is typically an active matrix or passive matrix liquid crystal display ("LCD") although other display technologies may be used as well. Graphics controller 135 can also be coupled to an optional external display or standalone monitor display 156 as shown in FIG. 3. One graphics controller that can be employed as graphics controller 135 is the Western Digital WD90C24A graphics controller.

4. Document ID: US 6487556 B1

L14: Entry 4 of 11 File: USPT Nov 26, 2002

DOCUMENT-IDENTIFIER: US 6487556 B1
**** See image for Certificate of Correction ****
TITLE: Method and system for providing an associative datastore within a data processing system

Detailed Description Text (6):
Referring to FIG. 2, a block diagram depicts a data processing system, which may be implemented as a server, such as server 104 in FIG. 1, in accordance to the present invention. Data processing system 200 may be a symmetric multiprocessor (SMP) system including a plurality of processors 202 and 204 connected to system bus 206. Alternatively, a single processor system may be employed. Also connected to system bus 206 is memory controller/cache 208, which provides an interface to local memory 209. I/O bus bridge 210 is connected to system bus 206 and provides an interface to I/O bus 212. Memory controller/cache 208 and I/O bus bridge 210 may be integrated

as depicted.

Detailed Description Text (8):
Additional PCI bus bridges 222 and 224 provide interfaces for additional PCI buses 226 and 228, from which additional modems or network adapters may be supported. In this manner, server 200 allows connections to multiple network computers. A memory-mapped graphics adapter 230 and hard disk 232 may also be connected to I/O bus 212 as depicted, either directly or indirectly.

Detailed Description Text (11):
With reference now to FIG. 3, a block diagram illustrates a data processing system in which the present invention may be implemented. Data processing system 300 is an example of a client computer. Data processing system 300 employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Micro Channel and ISA may be used. Processor 302 and main memory 304 are connected to PCI local bus 306 through PCI bridge 308. PCI bridge 308 also may include an integrated memory controller and cache memory for processor 302. Additional connections to PCI local bus 306 may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter 310, SCSI host bus adapter 312, and expansion bus interface 314 are connected to PCI local bus 306 by direct component connection. In contrast, audio adapter 316, graphics adapter 318, and audio/video (A/V) adapter 319 are connected to PCI local bus 306 by add-in boards inserted into expansion slots. Expansion bus interface 314 provides a connection for a keyboard and mouse adapter 320, modem 322, and additional memory 324. SCSI host bus adapter 312 provides a connection for hard disk drive 326, tape drive 328, and CD-ROM drive 330 in the depicted example. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors. Data processing system 300 could be connected through graphics adapter 318 to a computer display (not shown).

Detailed Description Text (19):
The objects that form the relationship hub are called Associations, as represented by Association 403. Associations are loosely typed as defined by the problem domain. Each Association manages 1 . . . N Roles where each Role is related to a Terminal.

5. Document ID: US 6449679 B2

L14: Entry 5 of 11 File: USPT Sep 10, 2002

DOCUMENT-IDENTIFIER: US 6449679 B2
TITLE: RAM controller interface device for RAM compatibility (memory translator hub)

Brief Summary Text (7):
RAMBUS DRAM (RDRAM) is a type of memory developed by RAMBUS, Inc., Mountain View, Calif. It is anticipated that RDRAM can transfer data at up to 600 MHZ. RDRAM is being used in place of video RAM (VRAM) in some graphics accelerator boards, but it is not expected to be used for the main memory of processing systems until 1999.

Detailed Description Text (9):
Referring to FIG. 1, a Prior Art processing system 100 is described. The system

h e b b g e e e f e fe ef b e

includes a processor 102 coupled to a host bus 104. The host bus allows communication with a second level cache 106 and memory controller 108. The memory controller can access either the cache memory or main memory 110. The controller also functions as a bridge between the host bus and a second bus 112, such as a peripheral component interconnect (PCI) bus. The PCI bus is a 64-bit bus which can run at clock speeds of 33 or 66 MHZ. This bridge is sometimes referred to as a north bridge. A second bridge 114 can be provided to allow communication with another bus 116, such as an industry standard bus (ISA). This bridge, sometimes referred to as a south bridge, also allows access to integrated drive electronics (IDE) ports, universal serial bus (USB), and other devices.

Detailed Description Text (13):
Because different SDRAM memories are currently available, the memory module interface device is not desired. These different SDRAM devices include, but are not limited to, SDR SDRAM, DDR SDRAM, and DDR SGRAM. The SDR SDRAM is a single data rate SDRAM that supports data transfers on one edge of each clock cycle. The DDR SDRAM is a double data rate SDRAM that supports data transfers on both edges of each clock cycle, effectively doubling the memory chip's data throughput. The DDR-SDRAM is also called SDRAM II. The DDR SGRAM is a double data rate synchronous graphics RAM. Each of these memories communicate using a row/column protocol and either low voltage transistor--transistor logic (LVTTL) or SSTL.sub.-- 2.

☐ 6. Document ID: US 6275499 B1

L14: Entry 6 of 11 File: USPT Aug 14, 2001

DOCUMENT-IDENTIFIER: US 6275499 B1
TITLE: OC3 delivery unit; unit controller

Detailed Description Text (49):
Unit controller 104 performs line/trunk processing functions, and controls administration and maintenance within delivery unit 10. I-links and E-links between unit controller 104 and BCM 101 carry only control data (in iPL subframes) because unit controller 104 does not access DS0 channels. Unit controller 104 communicates with other cards of delivery unit 10 via iPL subframes. A low-level maintenance bus (LLMB) between unit controller 104 and BCM 101 is used for resetting BCM 101 and for fault isolation. Unit controller 104 is connected to a line/trunk manager (LTM) 113 in switch control center 11b. Messages regarding call processing functions for delivery unit 10 are transported between unit controller 104 and LTM 113. Unit controller 104 also provides an ethernet connection via an ethernet hub 13 for communicating with OC-3 manager 14, which has a graphics user interface (GUI) 15. The OC-3 manager 14 is a "service unit", whose functions include connection setup and release, billing administration, and file management.

Detailed Description Text (167):
CPU card 192 has two reset inputs: a power-up reset and a hardware reset. The power-up reset is distributed on CPU card 192 to an expansion PCI connector and to memory controller. The hardware reset is distributed on CPU card 192 to the expansion PCI connector, the processor, the bridge/MPIC 192b, the memory controller the cache devices the control registers and miscellaneous FPGA, the FLASH devices, and to the bus arbiter. CPU card 192 also has a soft reset signal generated by a register in the host bridge/MPIC 192b that is distributed to the processor and the cache devices.

h e b b g e e e f e fe ef b e

Detailed Description Text (378):
To connect delivery unit 10 to the control structure of switching system 11, two types of communications media are used. An ethernet link connects unit controller 104 to the OC-3 manager 14 via an ethernet hub 13. Messages regarding subsystem administration and maintenance are transported on this ethernet link. A control bus connects unit controller 104 to the line/trunk manager (LTM) 113, and it connects MTXI 105 to PVP 112. Call processing information is transported on the unit controller/LTM link. Messages regarding path verification for switching matrix transport channels are transported on the MTXI/PVP link.

7. Document ID: US 6208273 B1

L14: Entry 7 of 11 File: USPT Mar 27, 2001

DOCUMENT-IDENTIFIER: US 6208273 B1
** See image for Certificate of Correction **
TITLE: System and method for performing scalable embedded parallel data compression

Brief Summary Text (6):
Graphical output data generated by the CPU is written to a graphical interface device for presentation on the display monitor. The graphical interface device may simply be a video graphics array (VGA) card, or the system may include a dedicated video processor or video acceleration card including separate video RAM (VRAM). In a computer system including a separate, dedicated video processor, the video processor includes graphics capabilities to reduce the workload of the main CPU. Modern prior art personal computer systems typically include a local bus video system based on the Peripheral Component Interconnect (PC) bus, the Advanced Graphics Port (AGP), or perhaps another local bus standard. The video subsystem is generally positioned on the local bus near the CPU to provide increased performance.

Brief Summary Text (9):
The CPU typically reads data from system memory across the local bus in a normal or non-compressed format, and then writes the processed data or graphical data back to the or local bus where the graphical interface device is situated. The graphical interface device in turn generates the appropriate video signals to drive the display monitor. It is noted that prior art computer architectures and operation typically do not perform data compression and/or decompression during the transfer between system memory and the CPU or between the system memory and the local I/O bus. Prior art computer architecture also does nothing to reduce the size of system memory required to run the required user applications or software operating system. In addition, software controlled compression and decompression algorithms typically controlled by the CPU for non-volatile memory reduction techniques can not be applied to real time applications that require high data rates such as audio, video, and graphics applications. Further, CPU software controlled compression and decompression algorithms put additional loads on the CPU and CPU cache subsystems.

Brief Summary Text (11):
Computer systems are being called upon to perform larger and more complex tasks that require increased computing power. In addition, modern software applications require computer systems with increased graphics capabilities. Modem software applications include graphical user interfaces (GUIs) which place increased burdens

h e b b g e e e f e fe ef b e

on the graphics capabilities of the computer system. Further, the increased prevalence of multimedia applications also demands computer systems with more powerful graphics capabilities. Therefore, a new system and method is desired to reduce the bandwidth requirements required by the computer system application and operating software. A new system and method is desired which provides increased system performance without specialty high speed memory devices or wider data I/O buses required in prior art computer system architectures.

Brief Summary Text (13):
The present invention includes parallel data compression and decompression technology, referred to as "MemoryF/X", designed for the reduction of data bandwidth and storage requirements and for compressing/decompressing data at a high rate. The MemoryF/X technology may be included in any of various devices, including a memory controller; memory modules; a processor or CPU; peripheral devices, such as a network interface card, modem, IDSN terminal adapter, ATM adapter, etc.; and network devices, such as routers, hubs, switches, bridges, etc., among others.

Brief Summary Text (17):
In a fifth embodiment, the present invention comprises a network device, such as a router, switch, bridge, network interface device, or hub, that includes the MemoryF/X technology of the present invention. The network device can thus transfer data in the network at increased speeds and/or with reduced bandwdith requirements.

Brief Summary Text (21):
Systems that require a minimum of DRAM memory but also require high bandwidth do not need to use multiple memory devices or specialty DRAM devices in a wider configuration to achieve the required bandwidth when the MemoryF/X technology is utilized. Thus, minimum memory configurations can be purchased that will still achieve the bandwidth required by high-end applications such as video and graphics.

Brief Summary Text (23):
Where the MemoryF/X Technology is included in a device, data transfers to and from the device can thus be in either two formats, these being compressed or normal (non-compressed). The MemoryF/X Technology may also include one or more lossy compression schemes for audio/video/graphics data. Thus compressed data from system I/O peripherals such as the non-volatile memory, floppy drive, or local area network (LAN) may be decompressed in the device and stored into memory or saved in the memory in compressed format. Thus, data can be saved in either a normal or compressed format, retrieved from the memory for CPU usage in a normal or compressed format, or transmitted and stored on a medium in a normal or compressed format

Brief Summary Text (37):
In one embodiment, the present invention comprises an improved system and method for performing parallel data compression and/or decompression. The system and method preferably uses a lossless data compression and decompression scheme. As noted above, the parallel data compression and decompression system and method may be comprised in any of various devices, including a system memory controller, a memory module, a CPU, a CPU cache controller, a peripheral device, or a network device, such as a router, bridge, network interface device, or hub, among other devices. The parallel data compression and decompression system and method may be used to provide a reduction of data bandwidth between various components in a computer system or enterprise. The present invention may reduce the bandwidth requirements while increasing the memory efficiency for almost all data types within the computer system.

Detailed Description Text (5):
U.S. Pat. No. 5,838,334 titled "Memory and Graphics Controller which Performs

h e b b g e e e f e fe ef b e

Pointer-Based Display List Video Refresh Operations", whose inventor is Thomas A. Dye, and which issued on Nov. 17, 1998, is hereby incorporated by reference in its entirety as though fully and completely set forth herein.

Detailed Description Text (8):
U.S. patent application Ser. No. 09/056,021 titled "Video/Graphics Controller Which Performs Pointer-Based Display List Video Refresh Operations" and filed Apr. 6, 1998, whose inventor is Thomas A. Dye, is hereby incorporated by reference in its entirety as though fully and completely set forth herein.

Detailed Description Text (10):
FIG. 1 illustrates a block diagram of a prior art computer system architecture. As shown, prior art computer architectures typically include a CPU 102 coupled to a cache system 104. The CPU 102 couples to the cache system 104 and couples to a local bus 106. A memory controller 108, referred to as North Bridge 108, is coupled to the local bus 106, and the memory controller 108 in turn couples to system memory 110. The graphics adapter 112 is typically coupled to a separate local expansion bus such as the peripheral component interface (PCI) bus or the Accelerated Graphics Port (AGP) bus. Thus the north-bridge memory controller 108 is coupled between the CPU 102 and the main system memory 110 wherein the north-bridge logic also couples to the local expansion bus where the graphics adapter 112 is situated. The graphics adapter 112 couples to frame buffer memory 114 which stores the video data, also referred to as pixel data, that is actually displayed on the display monitor. Modem prior art computer systems typically include between 1 to 8 Megabytes of video memory. An I/O subsystem controller 116 is shown coupled to the local bus 106. In computer systems which include a PCI bus, the I/O subsystem controller 116 typically is coupled to the PCI bus. The I/O subsystem controller 116 couples to a secondary input/output (I/O) bus 118. Various peripheral I/O devices are generally coupled to the I/O bus 18, including a non-volatile memory, e.g., hard disk 120, keyboard 122, mouse 124, and audio digital-to-analog converter (DAC) 238.

Detailed Description Text (12):
The CPU 102 accesses programs and data stored in the system memory 110 through the memory controller 108 and the local bus 106. In processing the program code and data, the CPU 102 generates instructions and data that are then provided over the local bus 106 and generally the PCI bus or AGP bus to the graphics adapter 112. The graphics adapter 112 receives graphical instructions or pixel data from the CPU 102 and generates pixel data that is stored in the frame buffer memory 114. The graphics adapter 112 generates the necessary video signals to drive the video display device (not shown) to display the pixel data that is stored in the frame buffer memory 114. When a window on the screen is updated or changed, the above process repeats whereby the CPU 102 reads data across the local bus 106 from the system memory 110 and then transfers data back across the local bus 106 and local expansion bus to the graphics adapter 112 and frame buffer memory 114.

Detailed Description Text (23):
The IMC 140 may also generate appropriate video signals for driving video display device 142. The IMC 140 may generate red, green, blue (RGB) signals as well as vertical and horizontal synchronization signals for generating images on the video display 142. Therefore, the integrated memory controller 140 may integrate memory controller and video and graphics controller capabilities into a single logical unit. This greatly reduces bus traffic and increases system performance. In one embodiment, the IMC 140 also generates appropriate data signals that are provided to Audio DAC 238 for audio presentation. Alternatively, the IMC 140 integrates audio processing and audio DAC capabilities and provides audio signal outputs that are provided directly to speakers.

Detailed Description Text (28):
The IMC 140 may also include a high level protocol for the graphical manipulation

of graphical data or video data which greatly reduces the amount of bus traffic required for video operations and thus greatly increases system performance. This high level protocol includes a display list based video refresh system and method whereby the movement of objects displayed on the video display device 142 does not necessarily require movement of pixel data in the system memory 110, but rather only requires the manipulation of display address pointers in a Display Refresh List, thus greatly increasing the performance of pixel bit block transfers, animation, and manipulation of 2D and 3D objects. For more information on the video/graphics operation of the IMC 140, please see U.S. Pat. No. 5,838,334. The IMC 140 also includes an improved system and method for rendering and displaying 3D objects.

Detailed Description Text (55):
The bus interface logic 202 couples to the memory control unit 220. The MemoryF/X technology preferably resides internal to the memory controller block 220. A control bus 201 connects all units to the local CPU interface 202. An execution engine 210 is coupled through the control bus 201 to the local CPU interface 202 and the memory interface 221 and the execution engine 210 also couples to the memory controller. Local bus 106 data and commands are routed through the local CPU interface to the control bus 201 which in turn is coupled to the execution engine 210, the memory interface 221, the graphics engine 212, the Peripheral I/O bus interface 234, the VDRL engine 240, a video input and format conversion unit 235 and finally the audio & modem subsystem 236. In addition the execution engine 210 is coupled to the main system memory 110 through the memory controller 220 and the memory interface 221.

Detailed Description Text (56):
The graphics engine 212 is also coupled to the main system memory 110 through the memory controller 220 and the memory interface 221. Thus, data is read and written for rasterization and pixel draw output by the graphics engine 212 with assistance for data transfer and efficiency by the memory controller 220. In addition, the other blocks are coupled under similar circumstances through the memory controller 220 and memory interface 221 to the system memory 110.

Detailed Description Text (57):
As shown in FIG. 6 the memory controller 220 transfers data between the system memory 110 and the requesting units. The requesting units include the execution engine 210, local CPU or RISC interface 202, audio and modem subsystem 236, Video I/O interface 235, VDRL engine 240, peripheral bus interface 234 and graphics engine 212. The requesting units will request the memory controller 220 for data transfer operations to the system memory 110 through the system memory interface 221. Each requesting unit may represent or utilize a different compression format, allowing higher memory efficiency. Thus, there are pluralities of data compression formats under control of the requesting units and supported by the memory controller block 220.

Detailed Description Text (65):
The L3 data cache size will determine the average number of clocks required to return data to the requesting units of the IMC 140. In the present embodiment, most recently used data is stored in a non-compressed format in the L3 data cache 291. For data that resides in the L3 data cache 291, no compression or decompression action is required by the parallel compression and decompression unit 251. Thus, a transaction request with an L3 data cache hit can return data with less latency than a transaction request that requires a main memory 110 transaction. The L3 data cache 291 typically contains only uncompressed data, although in alternate embodiments the L3 cache 291 may store most recently used data in a compressed format, or in a combination of compressed and non-compressed formats. Thus the L3 data cache 291 located in the memory controller 210 can return most recently used data without the normal latency delay associated with conventional memory controllers.

h e b b g e e e f e fe ef b e

Detailed Description Text (72):
FIG. 8 also illustrates the internal diagram of the switch logic 261. The switch 261 performs data format and address conversion as well as the arbitration of multiple requests from a plurality of other units in the IMC 140. The switch logic 261 includes a crossbar switch 502 that performs the selection of the current memory transaction request. This selection is performed by one of a plurality of arbitration methods with the intention to deliver data first to units that must operate real tine memory transactions. In the preferred embodiment, the order of priority for such requesting units is first the display refresh requests from the VDRL engine 240, followed by the Video I/O unit 235, the Audio and Modem 236, the Local CPU/RISC interface 202, the Graphics engine 212 and execution engine 210, followed by the Peripheral I/O bus interface 234. The priority order, block size, and request latency is software programmable by the interface driver software for the IMC 140. Thus, the system performance and memory transaction efficiency and/or response can be adjusted dynamically by software control executed by the interface drivers. Such interface software is preferably executed on the CPU 102 but alternatively can be executed by the execution engine 210.

Detailed Description Text (163):
Data input may originate in the YUV format (typically video) or the RGB format (typically graphics) and may also be combined with alpha for transparency effect. In the preferred embodiment, if the data to be compressed is in Red, Green and Blue format, data is converted to the proper data format of Y (luminance), Red and Blue or is left in YUV format if that is the original source format. During the source read process the data format is converted to the preferred format and a number of compare steps are performed on each block as indicated. The Y values of the block of 4.times.4 pixels during load are compared to the previous values for the maximum and minimum Y values of two pixels. Once found the associated R and G values are stored corresponding to such minimum and maximum Y values. Thus the maximum Y and minimum Y are determined for each block. As the data for each pixel is read the maximum and minimum Y are located, the associated R, B and Alpha values for the minimum and maximum Y pixels are also stored 770.

Detailed Description Text (166):
Due to the nature of the compression requirements the preferred embodiment introduces a new method to achieve high quality fixed or variable image and video compression ratios using a combination of both the lossy and lossless engines. The IMC 140 compresses multiple data types and formats as discussed previously in this disclosure. When image data is compressed with only a lossy algorithm, image data with high detail can be blurred or washed out. Prior art performs lossy compression on image data with discrete cosine transforms by conversion into the frequency domain. These practices are expensive due to the high bandwidth requirements of the real time transformation for video and graphics from the time domain to the frequency domain.

Detailed Description Text (178):
Referring to FIG. 21, in addition to the lossless format and lossy formats, the IMC 140 preferably contains further multiple compression and decompression formats for efficiency and optimization of bandwidth within the memory controller device. Data Source blocks 310, 320, 330, 340, and 350 represent the compression format of data that is read from system memory 110, written from the CPU 102, read from the non-volatile memory 120, read from the I/O system controller 116, or read from the internal graphics blocks within the IMC 140 device, or alternatively as in prior art FIG. 1, read from the PCI or AGP buses 107 to the IMC 140. Destination blocks 360, 370, 380, 390, 396, 300 represent the compression format of data that is written to system memory 110, or read by the CPU 102 (transferred to the CPU 102 in response to a CPU read), written to the non-volatile memory 120, written to the I/O system controller 116, written to internal graphics blocks within the IMC 140 device, or alternatively as in prior art FIG. 1, written to the PCI or AGP buses

h e b b g e e e f e fe ef b e

107 from the IMC 140. Therefore, blocks 310, 320, 330, 340, 350 are considered the data source formats where data flows into or is generated within the IMC. Blocks 360, 370, 380, 390, 396, and 300 are destination formats where data flows out of the IMC. It is noted that destination formats become source formats on subsequent accesses by the IMC 140. Thus a compression format may be referred to as source format/destination format.

Detailed Description Text (181):
One form of data in the preferred embodiment is video display refresh list (VDRL) data as described in U.S. Pat. No. 5,838,334, referenced above. VDRL data comprises commands and/or data for referencing pixel/video data on a span line basis, typically from various non-contiguous memory areas, for refresh of the display. VDRL compressed data is expected to be a long stream of start and stop pointers including various slopes and integer data. Such data must be compressed with the lossless compression and decompression process according to the present invention. The following VDRL context register fields in the graphics engine can be programmed to cause screen data to be written back to system memory as lossless compressed screen lines 390(or sub-lines) during VDRL execution:

Detailed Description Text (189):
When enabled, each screen line (or span line) that is rendered or displayed (based on processing one or more VDRL segments) is compressed independently (for each screen line, a new compression steam is started and closed) and written back to memory at the current byte offset into pDestTopLine. In addition, the graphics engine writes back a pointer to the compressed screen line at the current pointer offset into pDestTopLinePtr. The current offsets into pDestTopLine and pDestTopLinePtr are managed by the graphics engine. The compressed screen data 300 and corresponding pointer list can be referenced as a compressed window by a subsequent VDRL 309. Preferably the workspace associated with the compressed window includes the following fields used by the graphics engine to indirectly access the compressed screen data:

Detailed Description Text (199):
For each independent triangle, the 3D-triangle setup engine generates two lossless compressed static data blocks using standard linear compression 360: an execution static data block, and a graphics engine static data block. For a given 3D window or object, all static data is written starting at a particular base address (pTopStatic). Each static data block is compressed independently (for each static data block, a new compression stream is started and closed) and written back to memory at the current compressed block offset into pTopStatic. In addition, the 3D triangle setup engine writes back a pointer to the compressed static data block (pStatic) in the appropriate static pointer line bucket. The format of pStatic comprises the following fields: static data block pointer offset, static format (indicating whether the data is compressed or not), the number of compressed blocks associated with the execution static data block, and the number of compressed blocks associated with the graphics engine static data block. Note that the number of compressed blocks for each static data block type is used to instruct the decompression engine 550 how much data to decompress.

Detailed Description Text (201):
A 3D-DL comprises a 3-dimensional draw list for rendering a 3-D image into memory, or onto the display. For each 3D window line (or sub-line), the 3D execution engine generates a lossless compressed stream of a 3D-DL 304. Each 3D-DL line is compressed independently (i.e. for each 3DDL line, a new compression stream is started and closed) and the resulting compressed 3D-DL line 390 is written back to memory 110. It is not necessary for consecutive lines of 3D-DL to be contiguous in memory. In addition, the 3D execution engine of the IMC 140 may write back a 3D-DL pointer to the compressed 3D-DL line 390 at the current pointer offset into the 3D-DL pointer list (p3DDLPtr). The resulting compressed 3D-DL lines 390 and corresponding 3D-DL pointer list 304 is parsed and consumed by the 3D graphics

engine 212. The graphics engine 212 uses the following 3D-DL context register fields:

Detailed Description Text (216):
pTopTex is the base pointer to a compressed texture table. pTopTex is loaded into the graphics engine 212 on a per 3D window basis. opTex is an offset into pTopTex that provides the graphics engine 212 with a pointer to the first compressed texture sub-block (i.e., LOD0, sub-block 0) associated with the targeted texture. opTex is a field located in a group attribute data block, RenderState. RenderState contains attributes shared by groups of triangles. The group attribute data block pointer, pRenderState, is contained in each 3D-DL 304 segment. Using pTopTex, opTex, and all of the texture attributes and modifiers, one of the graphics engine's texture address generation engines determine which critical texture sub-blocks 380 (pLodBlk) to prefetch.

Detailed Description Text (222):
For special graphics, video, and audio data types 306, 308 and 310 the data types can be associated with a respective compression format to achieve optimal compression ratios for the system.

Detailed Description Text (227):
Display data 300 can also be compressed and is typically compressed in a lossless format that is linear complete span lines. During the refresh of video to the display, the display compressed span lines 300 which have not been modified by the 3D graphics engine 212 are decompressed for display on the respective display device span line.

Detailed Description Text (235):
Data types texture data 302, 3Ddraw lists 304, 2D-draw lists 306, Digital video draw lists 308, and Virtual (video) Display Refresh List 309 all represent the audio, video and graphics media formats of the IMC as referenced in U.S. Pat. No. 5,838,334.

Detailed Description Text (238):
As discussed further below, data from the CPU may be compressed and stored in linear address memory with variable block sizes. This data from the CPU may be unrelated to the graphics data, and may result from invalidation of cache lines or least recently used pages (LRU), or requested memory from a CPU-based application. In this embodiment the driver requesting compression will handle the memory allocation and directory function for both the compressed and uncompressed data.

Detailed Description Text (244):
As shown, the method in step 802 first receives uncompressed data. The data may be CPU application data, operating system data, graphics/video data, or other types of data The data may originate from any of the various requesting agents.

Detailed Description Text (248):
Where the requesting agent is used to determine the compression mode, the method determines who is the requesting agent and then determines the compression mode based on the requesting agent. For example, if the requesting agent is a CPU application or associated driver, then a lossless compression should be applied. If or the requesting agent is a video/graphics driver, then lossy compression may be applied.

Detailed Description Text (249):
Where the data type is used to determine the compression mode, the method examines the data type of the data and determines the compression mode based on the data type of the data. Using the example above, if the data comprises application data, the compression mode is determined to be lossless compression. If the data comprises video/graphics data, then the compression mode may be lossy compression.

In the preferred embodiment, the determination of the compression mode is preferably inherently based on data type of the data, and the use of address range or requesting agent in determining compression mode may be implicitly based on the data type being stored in the address range or originating from the requesting agent.

Detailed Description Text (274):
In addition, if the compression operation is not required for a plurality of requesting agents or block sizes, such as graphics frame buffer or depth and texture compression, the compression address translation table 2710 is not required in the alternate embodiment.

Detailed Description Text (302):
In the preferred embodiment, the compression block size, representing the input data block before compression, is dynamic and can be adjusted in size to reduce latency of operation. For example, the local bus interface 106 may compress with input blocks of 32 or 64 bytes while video 235 or graphics engine 212 may compress with input blocks of 256 or 512 bytes. In the preferred embodiment the power-on software will set default block sizes and compression data formats for each of the requesting units and for specific address ranges. Also, the preferred embodiment includes software control registers (not shown) that allow interface software drivers to dynamically adjust the compression block sizes for a plurality of system memory performance levels. Thus, by dynamically adjusting the compression block sizes based on one or more of the requesting agent, address range, or data type and format, latency can be minimized and overall efficiency improved.

☐ 8. Document ID: US 6188602 B1

L14: Entry 8 of 11 File: USPT Feb 13, 2001

DOCUMENT-IDENTIFIER: US 6188602 B1
TITLE: Mechanism to commit data to a memory device with read-only access

Brief Summary Text (7):
Intel Corporation's 82802 firmware hub currently uses flash memory with two levels of status where the flash memory cannot be overwritten: write-locked and locked-down. The locked-down state prevents further set or clear operations to the write-lock and read-lock bits of the flash memory and provides the most protection from unauthorized erasures or overwriting. It is not possible to go from locked-down status to unlocked status without first powering down or resetting the computer system, however. The write-lock bit has similar limitations, as it must be set to the desired protection state prior to starting a program or erase operation and is sampled only at the beginning of the operation.

Detailed Description Text (3):
FIG. 1 is a simplified block of computer system 100 with which the present invention for unlocking flash memory and then re-locking the flash memory once the desired operation is complete may be utilized. Computer system 100 includes one or more central processing units (CPU) 102 coupled to host bus 104 to communicate with memory controller 106, input/output (I/O) controller 108, and firmware hub 110. System management bus (SMB) devices 112, audio encoder/decoder (codec) unit 114, and integrated disk electronics (IDE) disk drives 116 are coupled via corresponding communication busses to I/O controller 108. Additional communication connections

couple a variety of other devices to I/O controller 108 including universal serial bus (USB) devices 118, one or more general purpose I/O (GPIO) registers 120, lower pin count (LPC) devices 122, and peripheral component interconnect (PCI) agents 124 and devices in PCI slots 126, as known in the art. Memory 128 is coupled for communication with memory controller 106, including cache memory and main memory control functions.

Detailed Description Text (5):
Memory controller 106, I/O controller 108, and firmware hub 110 form chipset 138 which is implemented, for example, in the 810E chipset manufactured by Intel Corporation of Santa Clara, Calif. In chipset 138, memory controller 106 includes built-in graphics processing technology and software drivers. System manageability bus (SMB) 140 allows monitoring of critical system parameters such as cooling fan speed, input voltages, and temperatures. I/O controller 108 employs accelerated controller hub architecture which connects directly to memory controller 106, audio codes 114, IDE drives 116, USB devices 118, and PCI devices 126, thereby providing high bandwidth data transfers among components in computer system 100. Firmware hub 110 stores system and video basic input/output systems (BIOS) 142, generates random numbers for security features, provides register-based read and write protection for code/data storage blocks, and includes a command user interface (CUI) for requesting access to locking, programming, and erasing options in firmware hub 110. The CUI also handles requests for data residing in status, identification, and block lock registers. BIOS 142 is implemented in flash memory in firmware hub 110 to support capabilities, such as plug and play, that automatically update system parameters and software when hardware is added or changed in computer system 100.

Detailed Description Text (6):
FIG. 2 shows I/O controller 108 coupled for communication with firmware hub (FWH) 110 and super I/O controller 136. Flash memory 212 in firmware hub 110 includes a locking scheme with three levels of locking status: locked, unlocked, and lock-down. The default status of all blocks in flash memory 212 upon power-up or reset is locked. Locked blocks are fully protected from alteration. Any program or erase operations attempted on a locked block will return an error. The status of a locked block can be changed to unlocked or lock-down using appropriate software commands. An unlocked block can be locked by writing a lock command sequence.

Detailed Description Text (9):
Several signals control access by I/O controller 108 to flash memory 212 in firmware hub 110. Firmware hub 110 receives input voltage supply V.sub.pp from the power supply (not shown) to control erasure and programming of flash memory 212 by controlling lock status. A write enable signal controls writes to a command register and memory array in flash memory 212. An output enable signal enables output from flash memory 212 during a read operation. A chip enable signal activates internal control logic, input buffers, decoders, and sense amplifiers (not shown) in firmware hub 110. Memory addresses and data input/output signals are communicated for storing received data and transmitting requested data between I/O controller 108 and firmware hub 110.

Detailed Description Text (10):
A PCI reset signal 214 is transmitted to I/O controller 108 through OR gate 216. The present invention includes firmware hub reset logic 218, which sets reset signal 214 that is transmitted to I/O controller 108 via GPIO port 220 and OR gate 216. This feature allows flash memory 212 to be taken out of locked-down status and updated without re-booting or powering down computer system 100.

Detailed Description Text (17):
Transitions between states 302 through 314 is shown in further detail in FIG. 3a. Powering on computer system 100 puts flash memory 212 in power on reset state 302. If adequate power is being supplied, flash memory 212 enters state 330 where flash memory 212 in firmware hub 110 is reset to read/write access. Flash memory 212 then

enters state 332 to check whether there is a valid RBU image to update BIOS 142. If a valid RBU image does not exist, flash memory 212 enters state 334 where BIOS 142 executes a power on self-test. Control is then transferred to state 336 where BIOS 142 puts all protected blocks in firmware hub 110 in locked status before passing control to computer system's 100 operating system and entering run state 304. If a valid RBU image exists, flash memory 212 enters state 338 where BIOS 142 updates firmware hub 110 with a new BIOS image and then enters state 302 to load the new image by resetting computer system 100.

Detailed Description Text (20):
When a system management interrupt (SMI) is requested, flash memory transitions from run state 304 to state SMI access state 312, by first transitioning through state 340 to verify the data and state 342 to unlock flash memory 212 by outputting a reset pulse to firmware hub 110. Once the SMI handler updates the requested information successfully, control transitions to state 344 to lock flash memory 212. If an error occurs in attempting to update the information, control transitions to state 346 where an error code is set and then to state 344 to lock flash memory 212. Control then transitions back to run state 304 with a completion code indicating whether the SMI access was successful or not.

Detailed Description Text (22):
When a request to update the flash memory program is received, control transitions from run state 304 to state 346 where FWH reset logic 218 (FIG. 2) generates a reset pulse on GPIO 220 (FIG. 2). Control transitions to state 314 and the program updates information in firmware hub 110. When the flash program is finished executing, control is passed to power on reset state 302.

Detailed Description Text (23):
Advantageously, the present invention provides a mechanism for writing to flash memory 212 without rebooting or powering down computer system 100. Flash memory 212 is placed in read only mode during the initial boot, or power up, process. When an application or operating system program requires write access to flash memory 212, a reset signal is output to GPIO port 220 by firmware hub logic 218. GPIO port 220 is coupled to reset circuitry for flash memory 212 through OR gate 216. Upon sensing the reset pulse, flash memory 212 is reset in read/write mode, thereby allowing information to be written to flash memory 212. When the update is completed, flash memory 212 is returned to locked status to prevent unauthorized access to flash memory 212.

Detailed Description Text (24):
While the invention has been described with respect to the embodiments and variations set forth above, these embodiments and variations are illustrative and the invention is not to be considered limited in scope to these embodiments and variations. For example, the present invention may be used to search and extract content from a wide variety of sites in addition to or instead of vendor sites. For example, flash memory may be included in several locations in computer system 100 to provide a nonvolatile storage media in addition to the flash memory in firmware hub 110. Such uses of flash memory include an ISA bus interface to flash memory embedded on a circuit board, add-in card, and/or in-line memory modules (both single in-line memory modules (SIMMS), and dual in-line memory modules (DIMMs)). Personal Computer Memory Card International Association (PCMCIA) interface allows memory expansion using flash memory PC cards. PCI bus 132 interfaces with flash memory on a PCI add-in card; and the IDE bus interfaces with advance technology attachment (ATA) flash memory. The present invention may be utilized in any system using suitable flash memory in any location. Accordingly, various other embodiments and modifications and improvements not described herein may be within the spirit and scope of the present invention, as defined by the following claims.

☐ 9. Document ID: US 6157976 A

L14: Entry 9 of 11 File: USPT Dec 5, 2000

DOCUMENT-IDENTIFIER: US 6157976 A
TITLE: PCI-PCI bridge and PCI-bus audio accelerator integrated circuit

Abstract Text (1):
A semiconductor device with an embedded PCI 2.1 compliant bridge provides expanded functionality as system-level implementations of a PCI-to-PCI bridge, and enhances the level of integration possible. The embedded PCI-to-PCI bridge allows the creation of multi-function, multimedia add-on cards supporting multiple devices. Multi-function, multimedia subsystems that provide audio, graphics, MPEG, etc., are mapped into a bridged-to PCI-bus that keeps such traffic off the main PCI-bus. The advantage for the system or add-in card vendor is that the various multimedia chips that are combined can come from different sources, providing an optimized and highly customized combination of functions.

Brief Summary Text (5):
However, PCI loading constraints limit the number of devices that can be supported directly on a system motherboard or through expansion slots, so PCI-to-PCI bridge chips have been developed by a number of major suppliers to increase the number of available system expansion slots. Embedding a bridge in a semiconductor device that also supplies other functionality provides a higher degree of integration and enables the creation of multi-function, multimedia expansion cards, where a single add-in card could support audio, graphics acceleration and video conferencing.

Detailed Description Text (4):
In one embodiment, the system controller 14 comprises an Advanced Micro-Devices (Sunnyvale, Calif.) AMD-640 system controller ("Northbridge") has a 64-bit Socket-7 interface, integrated writeback cache controller, system memory controller, and PCI bus controller. Such Socket-7 interface is optimized for the AMD-K6 processor, providing 3-1-1-1-1-1-1-1 transfer timing for both read and write transactions from PBSRAM at 66 MHz. (The number sequence refers to the CPU clock "t" cycles for each operation, i.e., 3-1-1-1 means the first data will be available at the third "t" when issue the operation, then the consequence data only need additional one "t" cycle, and so on.)

Detailed Description Text (6):
In another embodiment, the peripheral-bus controller 18 comprises an AMD-645 peripheral bus controller ("Southbridge"). The AMD-645 has an integrated ISA bus controller, enhanced master mode PCI EIDE controller with Ultra DMA/33 technology, an ACPI-compatible power management unit, a USB controller, a PS2-compatible keyboard/mouse controller, and a real-time clock (RTC) with extended 256-byte CMOS RAM. The on-chip EIDE controller has a dual-channel DMA engine capable of interlaced dual-channel commands. High-bandwidth PCI transfers are achieved by a sixteen double-word data FIFO with full scatter and gather capability. The integrated USB controller has a root hub with two ports having 18-level-deep data FIFOs and built-in physical layer transceivers.

10. Document ID: US 6138183 A

L14: Entry 10 of 11 File: USPT Oct 24, 2000

DOCUMENT-IDENTIFIER: US 6138183 A
TITLE: Transparent direct memory access

Brief Summary Text (6):
will improve the handling and implementation of ISA and PCI devices in a single computer. The new proposal partitions the motherboard, allowing system integrators greater flexibility in their choices of components such as core logic, PCI bridges, and graphics controllers. Because integrators will have more component options, its advocates say, Common Architecture should reduce overall manufacturing costs. Devices based on Common Architecture will emphasize the newer PCI bus, with the older ISA bus supported as an optional feature. While ISA legacy devices will be fully supported, physical-component connections will be routed through the PCI bus to eliminate sideband signals that straddle the chip set. The PCI bus will support distributed direct memory access (DMA) on the ISA side. Manufacturers will have the option of using sideband signals and extra chips on either side. Analysts are nonplused, pointing out that system vendors seem satisfied with current PCI/ISA architectures, bus standards, and core-logic chip sets.

Brief Summary Text (13):
As a baseline feature, the PC audio system has been overlooked for years. While the graphics market has been the recipient of broad and rapid innovation, the PC audio system has been neglected. All of this is about to change. As the audio system makes the jump from ISA to PCI, it is afforded access to a tremendous increase in bandwidth that translates into a host of new features and innovation. One new feature, wavetable synthesis, has really been around for several years. However, most traditional wavetable solutions were implemented in an extra synthesis chip and required a wavetable ROM to store sound samples. The latest PCI audio controller chips integrate the wavetable synthesis onto an audio controller chip. With access to PCI's bandwidth, sound samples can be stored in main memory and the wavetable ROM can be eliminated. The net result is less cost for the audio subsystem.

Detailed Description Text (3):
In one embodiment, the system controller 16 comprises an Advanced Micro-Devices (Sunnyvale, Calif.) AMD-640 system controller ("Northbridge"), has a 64-bit Socket-7 interface, integrated writeback cache controller, system memory controller, and PCI bus controller. Such Socket-7 interface is optimized for the AMD-K6 processor, providing 3-1-1-1-1-1-1-1 transfer timing for both read and write transactions from PBSRAM at sixty-six MHz. (The number sequence refers to the CPU clock "t" cycles for each operation, i.e., 3-1-1-1 means the first data will be available at the third "t" when issue the operation, then the consequence data only need additional one "t" cycle, and so on.)

Detailed Description Text (5):
In another embodiment, the peripheral-bus controller 26 comprises an AMD-645 peripheral bus controller ("Southbridge"). The AMD-645 has an integrated ISA bus controller, enhanced master mode PCI EIDE controller with Ultra DMA/33 technology, an ACPI-compatible power management unit, a USB controller, a PS2-compatible keyboard/mouse controller, and a real-time clock (RTC) with extended 256-byte CMOS RAM. The on-chip EIDE controller has a dual-channel DMA engine capable of interlaced dual-channel commands. High-bandwidth PCI transfers are achieved by a sixteen double-word data FIFO with full scatter and gather capability. The

h e b b g e e e f e fe ef b e

integrated USB controller has a root hub with two ports having 18-level-deep data FIFOs and built-in physical layer transceivers.

Detailed Description Text (32):
It is estimated that twenty percent of a modem CPU's capacity is wasted in producing 16-bit stereo sound at 44.1 kHz with the ISA bus-based audio. In contrast, embodiments of the present invention need less than one percent of a CPU's maximum available resources to deliver the same level of performance. The elimination of the ISA system bottleneck enables CPU resources to be used for other activities, including the support of graphics and audio pre-processing by MMX-enabled applications.

☐ 11. Document ID: US 5970069 A

L14: Entry 11 of 11 File: USPT Oct 19, 1999

DOCUMENT-IDENTIFIER: US 5970069 A
TITLE: Single chip remote access processor

Brief Summary Text (3):
Electronic data networks are becoming increasingly widespread for the communication of divergent types of data including computer coded text and graphics, voice and video. Such networks enable the interconnection of large numbers of computer work stations, telephone and television systems, video teleconferencing systems and other facilities over common data links or carriers.

Brief Summary Text (4):
Personal computers and computer work stations are typically interconnected by local area networks (LANs) such as Ethernet, Token Ring, DECNet and RS-232, whereas remote systems are interconnected by wide area networks (WANs) such as V.34, ISDN Basic Rate Interface (BRI), and Frame Relay (T1/E1 or fractional T1/E1) networks. LANs and WANs can be interconnected by devices known as hubs, bridges and routers in unlimited configurations.

Detailed Description Text (29):
The software executed by CPU 90 is typically stored in a flash read only memory (ROM) which is coupled to local memory interface 66. The software instructions are then moved to a dynamic random access memory (DRAM) as needed, which is also coupled to local memory interface 66. During an execution, instructions are transferred from the DRAM to instruction cache 92 through memory controller 110.

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
L13 and graphics	11

h e b b g e e e f e fe ef b e

Display Format: KWIC Change Format

Previous Page Next Page Go to Doc#

h e b b g e e e f e fe ef b e